|  |  |
| --- | --- |
| Traveling Salesman Problem  19a Coppa di Algoritmi | |
|  | |
| Studente |  |
| Davide Taurisano |  |
|  |
|  |
| Committente |
| Luca Gambardella  Marco Cinus |
| Corso di laurea | Modulo |
| Ingegneria Informatica | Algoritmi Avanzati |
| Anno |  |
| 2018/2019 |  |

|  |  |
| --- | --- |
| Data |  |
| 05/05/2019 |  |

[Problema 3](#_Toc7910160)

[Strutture dati 4](#_Toc7910161)

[Point 4](#_Toc7910162)

[Route 4](#_Toc7910163)

[Main 4](#_Toc7910164)

[Soluzioni Implementate 5](#_Toc7910165)

[Algoritmo Costruttivo 5](#_Toc7910166)

[Algoritmo di ottimizzazione locale 5](#_Toc7910167)

[Algoritmo meta-euristico 6](#_Toc7910168)

[Stop condition 6](#_Toc7910169)

[Software deterministico 7](#_Toc7910170)

[Random in Main 7](#_Toc7910171)

[Numero di iterazioni 7](#_Toc7910172)

[Scrittura su file 8](#_Toc7910173)

[Esecuzione del programma 8](#_Toc7910174)

[Run Automatizzate 9](#_Toc7910175)

[Risultati ottenuti 10](#_Toc7910176)

[Conclusioni 11](#_Toc7910177)

Problema

Il problema del commesso viaggiatore consiste nella ricerca del ciclo hamiltoniano più breve

all’interno di un grafo pesato completo. Questo tipo di problema appartiene alla classe

dei problemi NP-Completi.

In Figura 1 è mostrato uno dei problemi TSP con soluzione,

sottoinsieme dei problemi NP.

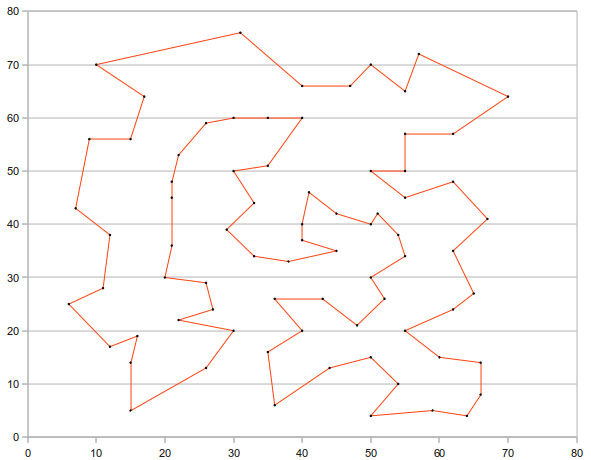


Figura 1: Eil76, problema di TSP da 76 città

Il compito assegnato è di generare una soluzione ammissibile e quanto più possibile

vicina a quella ottimale (ovvero la migliore) in 3 minuti di esecuzione del programma. Il

programma deve poter lavorare su 10 problemi diversi:

ch130;

d198;

eil76;

fl1577;

kroA100;

lin318;

pcb442;

pr439;

rat783;

u10160.

Strutture dati

## Point

La classe point è formata dai campi **x** ed **y**, rappresenta una città, quindi rispettivamente le sue coordinate.

Contiene il metodo **euclideanDistance(Point a,Point b)**, che permette di calcolare la distanza tra due città e viene utilizzato inizialmente per costruire la matrice delle distanze

## Route

Route è la classe che rappresenta un tour, **nel quale la prima città è salvata due volte, all’inizio e alla fine**.

Contiene il metodo per calcolare la distanza del tour che utilizza a sua volta la matrice delle distanze e contiene il metodo per scrivere il tour su file.

## Main

Main è la classe principale del programma, contiene informazioni statiche importanti come la matrice delle distanze, la variabile di random ed il tempo di inizio del programma.

E’ la classe a cui passiamo i parametri iniziali:

* Nome del file tsp di input
* [Seed]
* [Numero di iterazioni]

Soluzioni Implementate

## Algoritmo Costruttivo

Come algoritmo costruttivo è stato scelto di implementare un **Nearest Neighbor.**

E’ stato utile solo per costruire un tour iniziale da dare in input al prossimo algoritmo, ma non ha avuto un ruolo significativo in termine di migliorie del risultato finale.

Scegliere un algoritmo che genera un tour a random non avrebbe né peggiorato né migliorato la situazione.

La classe nella quale è stato implementato si chiama **NearestNeighbor**, si avvia chiamando il metodo statico **nearestNeighbor** e passando come parametro l’array di nodi (Città) lette nel file di input e ritorna una Route il primo tour costruito.

## Algoritmo di ottimizzazione locale

Come algoritmo di ottimizzazione locale è stato scelto di implementare un classico **2-Opt.**

Ovviamente ad ogni iterazione del for non viene effettuato uno scambio effettivo dei nodi per poi calcolare la distanza del tour e verificarne la miglioria, ma viene chiamata una funzione che stima il guadagno che si potrebbe avere effettuando uno swap tra dei determinati nodi.

La classe nella quale è stato implementato si chiama **TwoOpt**, si passa come parametro l’array di nodi (Città) ottenute dall’algoritmo precedente, si avvia chiamando il metodo **start** che ritorna una Route contenente il nuovo tour ottimizzato.

## Algoritmo meta-euristico

Come algoritmo meta-euristico è stato scelto di implementare un **Simulated Annealing.**

Riceve in input il tour ottimizzato dal TwoOpt ed inizia la sua ottimizzazione una volta chiamato ed eseguito il metodo **start.**

La temperatura ed l’alpha vengono generate in modalità random, con valori rispettivamente pari tra il 100 e 200 per la prima e tra 0.90 e 0.97 per la seconda.

Per “mischiare” il tour, quindi per uscire dai minimi locali è stato scelto di implementare un **double bridge**.

L’algoritmo ritorna la Route con il tour ottimizzato ed il numero di volte che è stato eseguito il for interno al while (for da 100 iterazioni).

### Stop condition

La stop condition dell’algoritmo viene raggiunta in 3 casi:

* Si è superata la soglia del tempo massimo da quando il programma è iniziato, in questo programma la soglia massima è stata settata a 150 secondi, lasciando 30 secondi di tempo tampone per contenere eventuali ritardi e tempi di scrittura su file
* E’ stata raggiunta la distanza minima ottenibile su quella mappa di città data in input. quindi è stato ottenuto il risultato ottimale
* Nella versione finale del software, quando si inseriranno i seed per ottenere la soluzione, si passeranno anche il numero di volte in cui viene eseguito il for loop all’interno del while, precedentemente salvato quando era stato salvato il valore del seed.

Software deterministico

Il software deve garantire un risultato deterministico, quindi con gli stessi input si devono ottenere gli stessi output

## Random in Main

L’unico Random utilizzato in tutto il programma è quello statico contenuto nel Main, al quale viene assegnato come seed, il currentMillis durante la fase di ricerca dei seeds migliori ed invece il seed passato a parametro nella fase di testing del programma

## Numero di iterazioni

Basare il proprio algoritmo su una durata di n secondi, non garantisce un risultato deterministico, perché le prestazioni del pc in quella finestra di tempo potrebbero variare a seconda del carico di lavoro. Quindi ottenendo un risultato migliore o peggiore, comunque diverso.

Per bypassare questo problema, si passa oltre al seed, il numero di iterazioni che deve eseguire il for loop all’interno del while, così basando la durata del programma non su un intervallo di tempo (Nel quale il numero di iterazioni potrebbe variare) ma su un numero definito di iterazioni indipendenti dal tempo.

Queste saranno calcolate e calibrate durante la fase di ricerca dei seed, in modo da rientrare nella finestra di tempo massima di 3 minuti.

Scrittura su file

La scrittura su file viene effettuata dal metodo **printToFileTour(String path,String filename,int length)** della classe **Route.**

Il file.tour è nel formato che permette di testarlo attraverso lo script **TspTourChecker.py**

Il file di output avrà il seguente nome:

**OPT\_*nomefiletspinput.tsp*.tour**

es. con file u1060.tsp:

**OPT\_u1060.tsp.tour**

Esecuzione del programma

Il programma viene eseguito e testato sulla macchina fornita dal docente.

Il programma verrà eseguito con **mvn test,** quindi è già stata preparata la classe di test fornita dal docente con i metodi di test necessari per ogni file tsp di input.

Durante le fasi di ricerca seed e testing è stato generato il jar richiamabile con i seguenti modi:

* Fase di ricerca seed  
  **java -jar AlgoritmiProgetto.jar *nomefileinput.tsp***
* Testing  
  **java -jar AlgoritmiProgetto.jar *nomefileinput.tsp* [seed] [numeroiterazioni]**

Run Automatizzate

Sono stati creati due script in python, uno per automatizzare la ricerca dei seed (findBest.py) ed uno per testarli (testSeed.py)

**findBest.py**

import subprocess

files=["fl1577.tsp","lin318.tsp","pcb442.tsp","rat783.tsp","u1060.tsp"]

import subprocess

while True:

for filename in files:

subprocess.call(['java', '-jar', 'AlgoritmiProgetto.jar',filename])

**testSeed.py**

files=["ch130.tsp","d198.tsp","eil76.tsp","fl1577.tsp","kroA100.tsp","lin318.tsp","pcb442.tsp","pr439.tsp","rat783.tsp","u1060.tsp"]

seeds=["1556838468638","1556838308988","1556838319270","1556838323370","1556838335182","1556838345451","1556838355700","1556838541092","1556838551522","1556838561812"]

iterations=["549","234","1633","2","870","77","35","38","9","4"]

import subprocess

while True:

i=0

for filename in files:

seed=seeds[i]

iteration=iterations[i]

subprocess.call(['java', '-jar', 'AlgoritmiProgetto.jar',filename,seed,iteration])

i=i+1

Risultati ottenuti

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem** | **Best Known** | **Student Result** | **Error** |
| ch130 | 6110 |  | -1 |
| d198 | 15780 |  | -1 |
| eil76 | 538 |  | -1 |
| fl1577 | 22249 |  | -1 |
| kroA100 | 21282 |  | -1 |
| lin318 | 42029 |  | -1 |
| pcb442 | 50778 |  | -1 |
| pr439 | 107217 |  | -1 |
| rat783 | 8806 |  | -1 |
| u1060 | 224094 |  | -1 |
|  |  |  |  |
|  |  |  |  |
| **FINAL RESULT** | |  | -1 |

Conclusioni

Personalmente il progetto mi è piaciuto, sono contento dei risultati ottenuti, pur avendo utilizzato degli algoritmi già spiegati in classe.

Ho trovato difficoltà nel rendere il problema deterministico, ma sono riuscito a trovare una soluzione, probabilmente non ottimale, però mi ha comunque soddisfatto.